LEGO® TC logo

LEGO Dacta
Educational Products
555 Taylor Road
Enfield, CT 06082

LEGO Australia Pty. Ltd.
P.O. Box 639
Lane Cove
N.S.W. 2066

ISBN 0-914831-68-2
198323
Printed in U.S.A.

LEGO® TC logo

Getting Started

Getting Started

No part of the documents contained herein may be reproduced, stored in retrieval systems or transmitted, in any form or by any means, photocopying, electronic, mechanical, recording or otherwise, without the prior approval in writing from the LEGO Group.

2nd revision, January, 1989

ISBN 0-914831-68-2

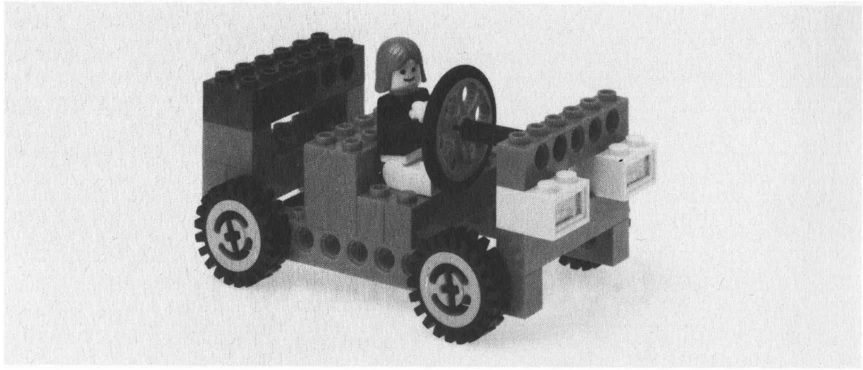
LEGO® and the LEGO logo® are registered trademarks of INTERLEGO AG. ©1987, 1989 LEGO Group.

Apple and the Apple logo are registered trademarks of Apple Computer, Inc.

IBM is a registered trademark of International Business Machines Corporation.

MS-DOS is a registered trademark of Microsoft Corporation.

Project 1: Soap Box Derby



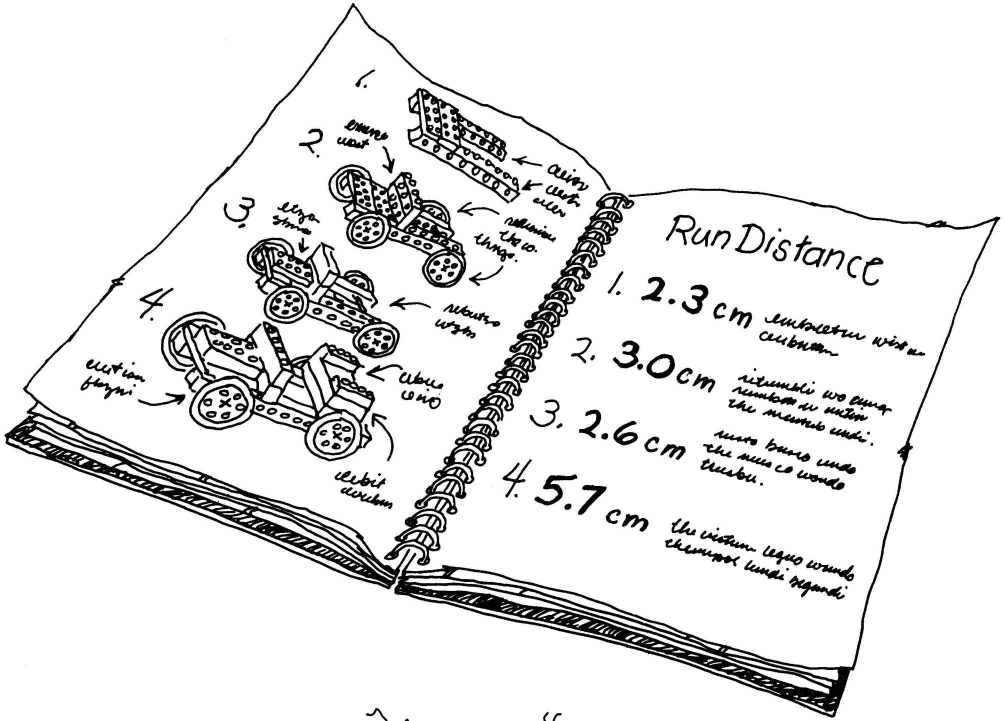
Build a LEGO® car to race in the Soap Box Derby!

There are many different ways to build a LEGO® car. Some examples are shown on the back page of Activity Card 4.

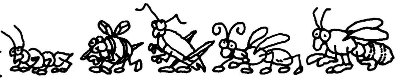
Racing the Car

Start your car at the top of the ramp and release it gently. Mark the spot where your car stops.

Race your car down the ramp a few times. Measure how far your car goes each time, and record the results in your Inventor's Notebook or on your worksheet. Calculate the average distance for your car.



BUG BOX



An axle isn't spinning freely.
Something is dragging.
You forgot the tires!

BIG?
HEAVY
?
WHY?
LIGHT?
LITTLE?
HOW!?

Making a Theory

There are many **factors** that could influence how far your car goes: The weight of the car, the number of wheels, the size of the wheels.

Which factors should you change to make the car go further? Should the car be heavier? Lighter? Should it have more wheels? Bigger wheels?

What is your **theory** of what makes the car go further?

Make a change and record the new results in your Inventor's Notebook. Change only one factor at a time. That way, you can tell which factors make a difference in the distance your car moves.

Challenges

1. What will happen if you make the ramp steeper? Raise the end of the ramp and try it!

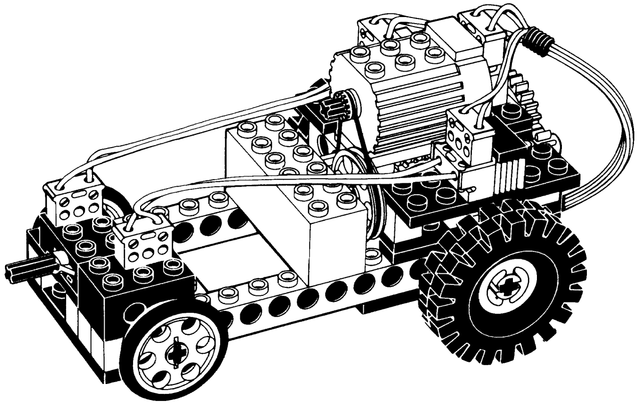
2. Imagine that you didn't have any rulers in the classroom. How could you keep track of how far your car goes?

Note: You will be using your soap box car for other activities in this booklet so you may want to keep it assembled.

Project 2: Motorized Car

To start a car moving, someone or something must supply a **force**. When the car rolls down a hill, the force comes from **gravity**. When you push the car with your hand, **you** supply a force.

Another way to supply a force is with a **motor**. Build the motorized car on Activity Card 3. You will use this model in activities throughout this book so keep the car assembled.



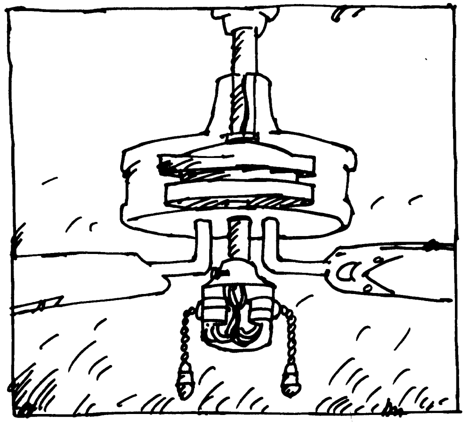
The motor is connected to the rear axle of the car. So when the motor turns, the rear axle will turn — and the car moves.

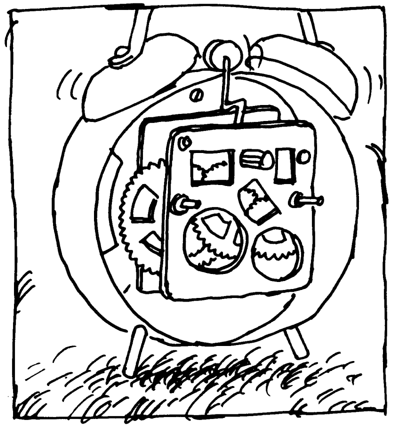
The model uses a ''small-to-big'' connection. The pulley on the motor is smaller than the pulley on the axle. The axle turns **more slowly** than the motor, but with **greater force**. Trace the motor, pulley and gear connections on your car.

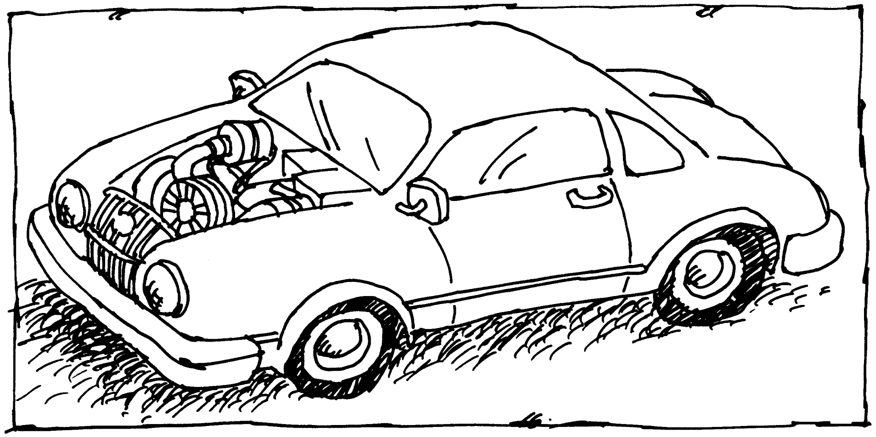
Motors

There are many types of motors. Some motors, like the motors in real cars, get their power by burning **gasoline** or other fuels. Other motors, like the LEGO® motors, use **electricity.**

There are motors in many types of machines. There are motors in cars, in washing machines, in clocks, in merry-go-rounds.





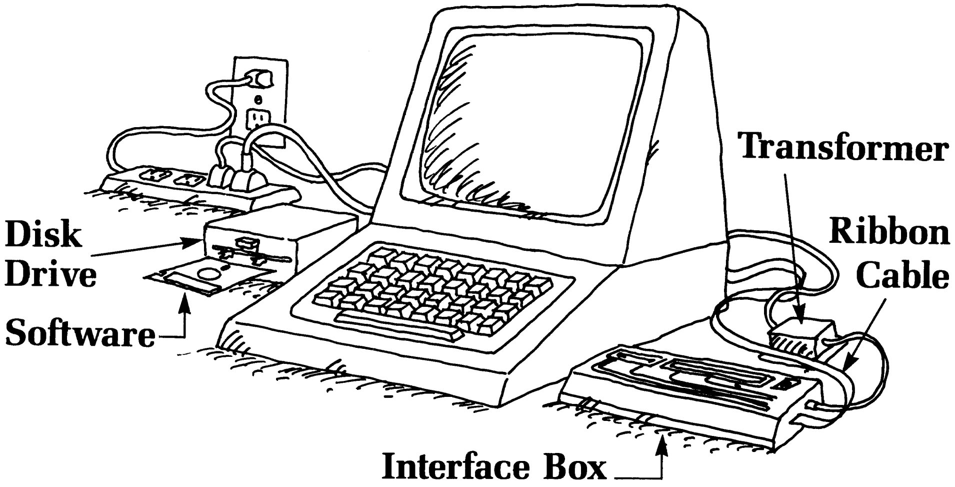


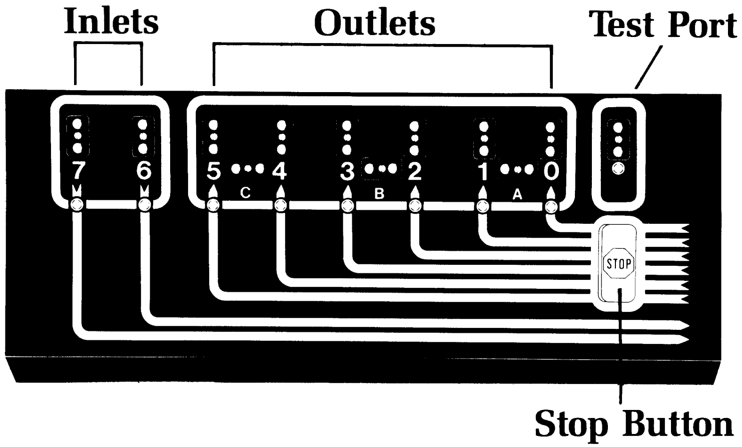
The LEGO® TC logo System

Using the computer, you can **program** the things you build out of LEGO® elements. You can program a car to move forward and back. You can program a traffic light to turn on its red light, its green light, then its yellow light — then start over again!

The LEGO® TC logo system works somewhat like the human body. The computer is like the brain. Your brain sends messages to your muscles, telling them how to move. Similarly, the computer sends messages to LEGO® motors and lights, telling them to turn "on" or "off."

Other LEGO® pieces, known as **sensors**, send messages back to the computer. A touch sensor, for example, is like a finger. If it is being pressed, it sends a message that says "true." If it is not being pressed, it sends a message that says "false." You will learn more about sensors later in this book.





The computer communicates with LEGO® pieces through the black **interface box**. The interface box has 12 sets of holes called **ports**. Some of the ports, called outlets, are used to **send** messages (like "on" or "off") to motors and lights. Other ports, called inlets, are used to **receive** messages (like "true" or "false") from sensors.

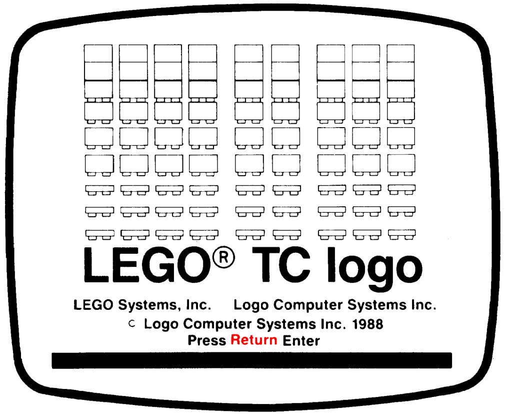
The red "stop button" lets you stop motors quickly. When you press the button, you cut off power to all outlet ports, so all motors and lights will turn off.

The test port supplies power at all times (even when the "stop button" is pushed). You can use it to test LEGO® machines. Use a LEGO® wire to connect your car to the test port. The car should begin moving. If the car does not run smoothly, look for places where pieces are rubbing against one another.

Using the Computer

To use the computer, you need the LEGO® TC logo software disk. Different computers start the LEGO® TC logo software in different ways. Look at the *Setup Guide* to find out how to start the LEGO® TC logo software on your computer.

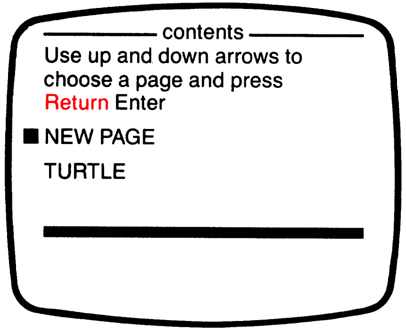
Once the LEGO® TC logo software is loaded, the computer screen will look like this:



LEGO® TC logo

LEGO Systems, Inc. Logo Computer Systems Inc.
© Logo Computer Systems Inc. 1988
Press Return Enter

Press the Return Enter key. (On IBM computers, Enter looks like this: ↵ .) Then wait until you see the Contents menu.



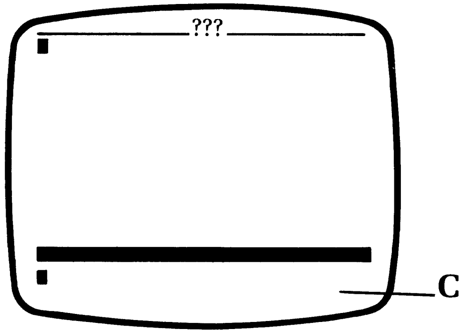
contents
Use up and down arrows to
choose a page and press
Return Enter

■ NEW PAGE
TURTLE

The LEGO® TC logo disk is organized into **pages**. The Contents menu lists all the pages on the disk.

The blinking light is called the **cursor**. You can move the cursor by pressing the **up arrow** or **down arrow** keys.

Move the cursor next to the words NEW PAGE. Then press the Return Enter key. A new page will appear:



???

Command Center

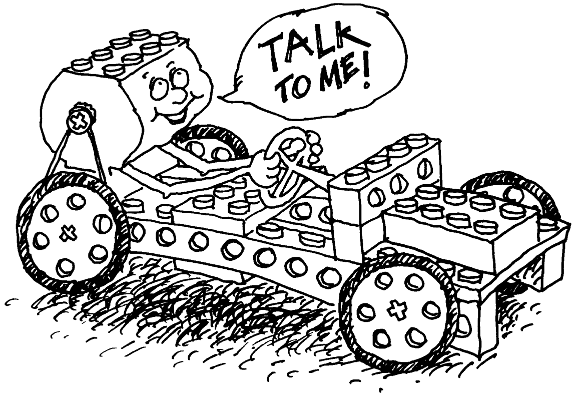
The screen is divided into two parts. The bottom part of the screen is called the **Command Center**. This is where you type instructions to the computer.

Computer Control

Plug the motor of your car into port A. Then type each of these LEGO® TC logo commands into the Command Center. After each command, press the Return Enter key.

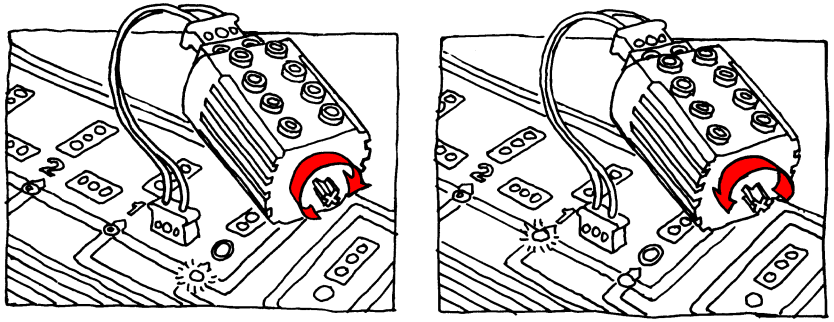
Command	What the computer does
`talkto "A`	Gets ready to "talk to" port A.
`on`	Turns on motor
`rd`	**R**everses **d**irection of motor
`off`	Turns off motor
`onfor 10`	Turns on motor for 1 second, then off
`cc`	**C**lears the **C**ommand Center

Spaces and quotes are very important. In `talkto "A` you must leave a space between `talkto` and `"A`. But you must **not** leave a space between `talk` and `to`, or between the quotes (`"`) and `A`.



You can put more than one command on a line. For example, type `onfor 32 rd onfor 27`, then press the Return Enter key.

Notice how the red lights on the interface box indicate which way the motor is turning!



Challenges

1. How far does your car go when you type `onfor 20`?

2. What should you type if you want your car to go forward one meter?

3. Write several commands on one line to make your car go forward one meter, then return to its starting point.

BUG BOX

There is friction in the wheels or axles.
The wire isn't plugged into the motor.
The wire isn't plugged into port A.
You didn't put a number after `onfor`.
The stop button is pushed in.
You forgot the quotes in `"A`.
You typed `"A"` instead of `"A`.

Talking to Other Ports

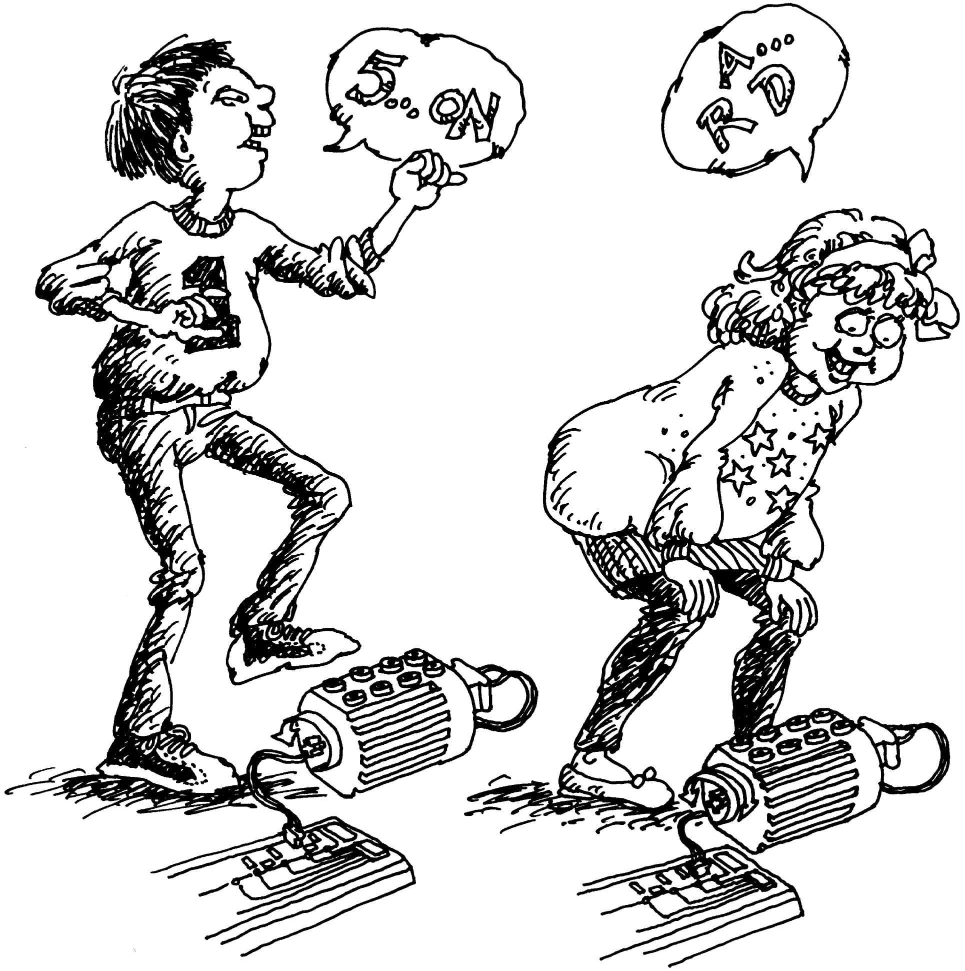
The computer can give instructions to any of the outlet ports. You must simply tell the computer which port it should "talk to."

Plug one motor into port A and another motor into port C. Then try each of these commands. Remember to press the Return Enter key after each command.

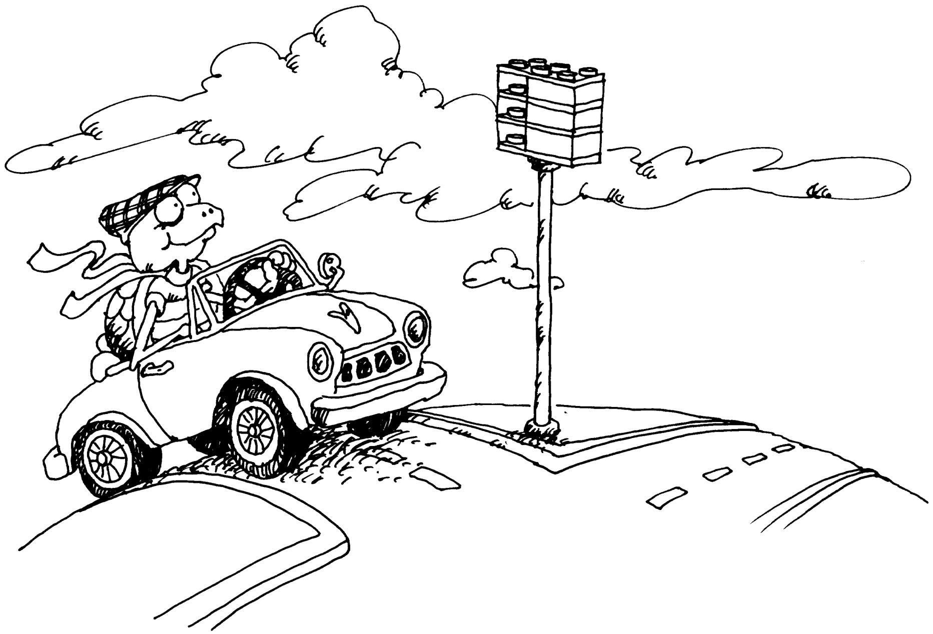
Command	What the computer does
`talkto "C`	Gets ready to "talk to" port C
`on`	Turns on motor in port C
`rd`	Reverses direction of motor in port C
`talkto "A`	Gets ready to "talk to" port A
`onfor 10`	Turns on motor in port A for one second
`talkto "C`	Gets ready to "talk to" port C
`off`	Turns off motor in port C

You can use the abbreviation `tto` instead of `talkto`. So `tto "A` is the same as `talkto "A`.

When a motor is in port A, port B, or port C, you can control whether it is **on** or **off** and you can control its **direction**. When a motor is in an outlet port with a number (0,1,2,3,4, or 5), you can only control whether it is **on** or **off**.



Project 3: Traffic Lights



Traffic lights coordinate the movement of cars and people. They use colors as codes. Red is the code for Stop. Green is the code for Go.

You can build a LEGO® traffic light and program the computer to control the lights. See Activity Card 2 for instructions on how to build the light.

Controlling the Lights

Plug the red light into port 0, the yellow light into port 1, and the green light into port 2.

Then try these commands. Remember to press the Return Enter key after each one.

`talkto 0`	Talk to port 0
`on`	Turn on the red light
`off`	Turn off the red light
`talkto [0 1 2]`	Talk to all three ports
`on`	Turn on all three lights
`off`	Turn off all three lights

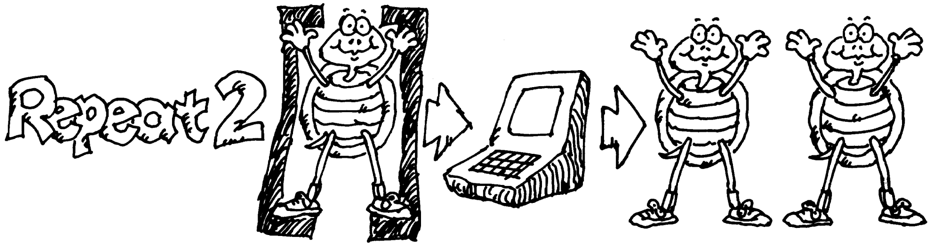
Notice that you can talk to more than one port at a time. Simply put the names of all the ports you want inside **square brackets**, like `[0 1 2]`. This is called a **list**.

You can plug the lights into any of the “outlet” ports. Just make sure that you are “talking to” the right port!

Challenge

Make the green light go on for five seconds.

Repeating and Repeating and …



You can tell the computer to **repeat** a sequence of commands. For example, try this:

```
repeat 2 [talkto 0 onfor 20 talkto 2
onfor 20]
```

The computer will repeat everything that is inside the square brackets (that is, everything in the **list**). The traffic light will go Red-for-two-Green-for-two, then again Red-for-two-Green-for-two.

You can use `repeat` to make a flashing light. Try this:

```
talkto 1
repeat 8 [onfor 10 wait 10]
```

The command `wait 10` tells the computer to wait for one second. So the computer tells the yellow light to go on for one second, then the computer waits for one second (while the light is off). Then the computer repeats the cycle seven more times: on-for-one-off-for-one, on-for-one-off-for-one, . . .

Here's another way to make the light flash. Type `flash 20 10`, and press Return Enter. The light will go on for two seconds, off for one second, over and over. Type `off` to stop the flashing.

The `flash` approach is somewhat different from the `repeat` approach. When you use `flash`, the light will continue to flash "forever" (until you give a new command to the light). Also, you can enter other commands while the light is flashing. For example, try:

`talkto 1`	◀ Talk to the yellow light
`flash 30 5`	◀ Start the light flashing
`talkto 0`	◀ Talk to the red light
`onfor 40`	◀ Turn on the light for four seconds
`talkto 1`	◀ Talk to the yellow light
`off`	◀ Stop the flashing

Challenges

1. Use `repeat` to make the lights work like a real traffic light: Red, then Green, then Yellow, then Red, then Green, then Yellow.

2. Try changing each of the numbers in
`repeat 8 [onfor 10 wait 10]`
What happens for each change? What happens if you change more than one of the numbers?

3. Make the red light flash quickly (on for a half second, off for a half second), while the green light flashes slowly (on for 3 seconds, off for 3 seconds).

Defining New Words

The computer already understands quite a few commands. It understands `on`, `off`, `talkto`, `repeat` and about 100 others.

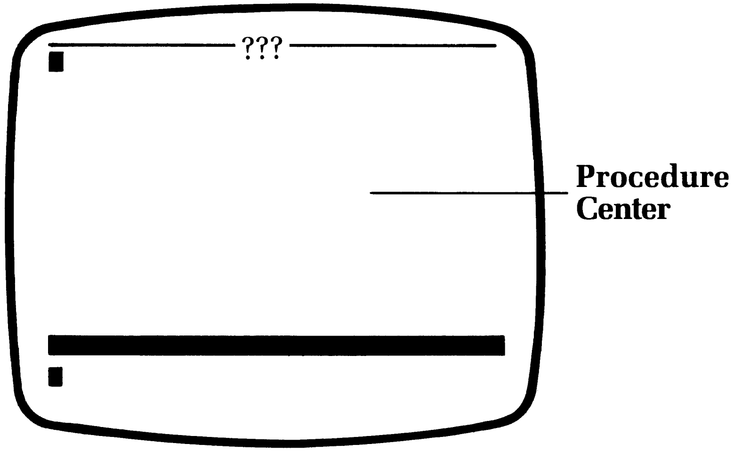
That might seem like a lot of commands. But it is often useful to teach the computer **new** commands.

Let's say that you want the red light to go on for three seconds. You could do this by typing `talkto 0 onfor 30`. But wouldn't it be nice if all you had to do was type the word `red`?

To do that, you have to create a **procedure** for `red`. That is, you have to tell the computer exactly what `red` means.

To define new procedures, you must move the cursor to the Procedure Center at the top of the screen. To move the cursor **up**, hold down the Apple key () and press the **U** key (without letting go of the key). On IBM computers hold down the Control (Ctrl) key and press the **U** key.

Now that you're at the top of the page, you can write a procedure.



Type the definition for `red`:

```
to red
talkto 0
onfor 30
end
```

◀ Procedures must start with `to`

◀ Procedures must end with `end`

Now that you have defined the procedure, you can use `red` just as you would use any of the "built-in" commands like `on` and `talkto`. (The built-in commands are also procedures. Sometimes they are called **primitive procedures** or just **primitives**.)

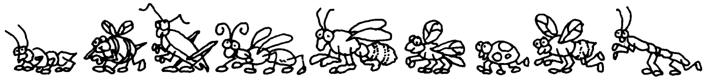
To "run" the new procedure, you must return to the Command Center. To move the cursor **down** to the Command Center, hold down the Ctrl key and press the **D** key.

In the Command Center, type `red` (and press Return Enter). What happens?

If the procedure did not work, you probably made a typing mistake in the procedure definition. Go back **up** to the Procedure Center (-U Ctrl-U) and look for the mistake. This process is called **debugging** the procedure.

You can give a procedure any name you want, as long as you don't use one of the built-in primitive names. A procedure name must be a single word. You can't use `red light`. But `red.light` and `red_light` count as single words. It is a good idea to give your procedures meaningful names, so that you can remember what they do.

BUG BOX



You forgot the space between `to` and `red`.
You used a hyphen (`-`) in the name of a procedure.
You typed `to red` (instead of `red`) in the Command Center.

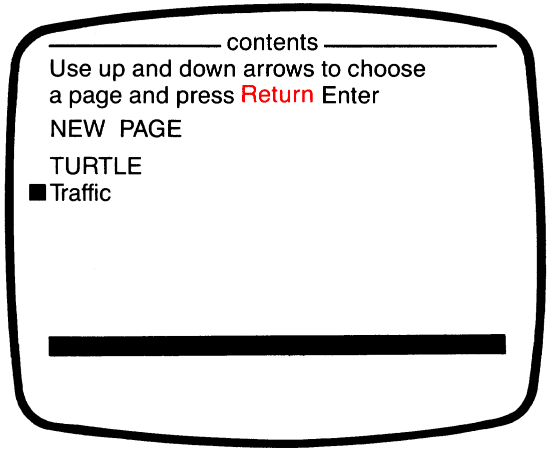
Naming and Saving Pages

Now that you have a procedure on your page, you might want to save the page. That way, you can use the procedure again later.

Before saving the page, you must give the page a name. To name the page **traffic**, type `namepage "traffic` in the Command Center. The new name should appear at the top of the screen! You can use any name you want but don't forget the quotes. (Instead of typing `namepage`, you can simply type `np`.)

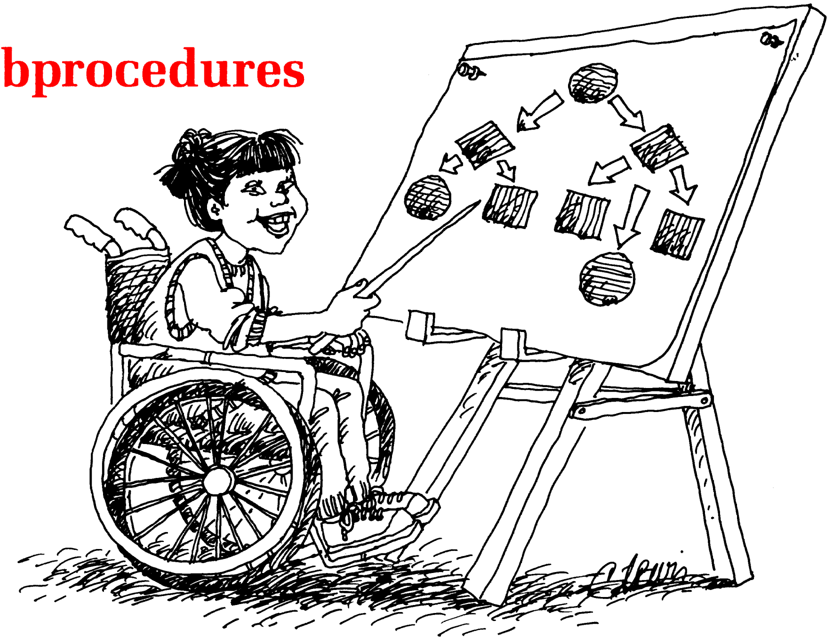
To save the page, simply press the Escape (**Esc**) key. The computer returns to the `Contents` menu, and it automatically saves your page.

To return to the `traffic` page, move the cursor to the word `traffic`. (Use the ↓ key.) Then press the Return Enter key.



```
contents
Use up and down arrows to choose
a page and press Return Enter
NEW PAGE
TURTLE
■Traffic
```

Subprocedures



New procedures can be used just like the "built-in" primitives. For example, you can use the new procedures in the definitions of other new procedures.

Go back to the Procedure Center (press -U Ctrl-U) and add two new procedures:

```
to green
talkto 2
onfor 30
end

to lights
red
green
end
```

`lights` uses two other procedures in its definition. When procedures (like `red` and `green`) are used inside other procedures, they are called **subprocedures.**

Now go back to the Command Center (press -D Ctrl-D) and type `lights`. Press Return Enter. Does it work?

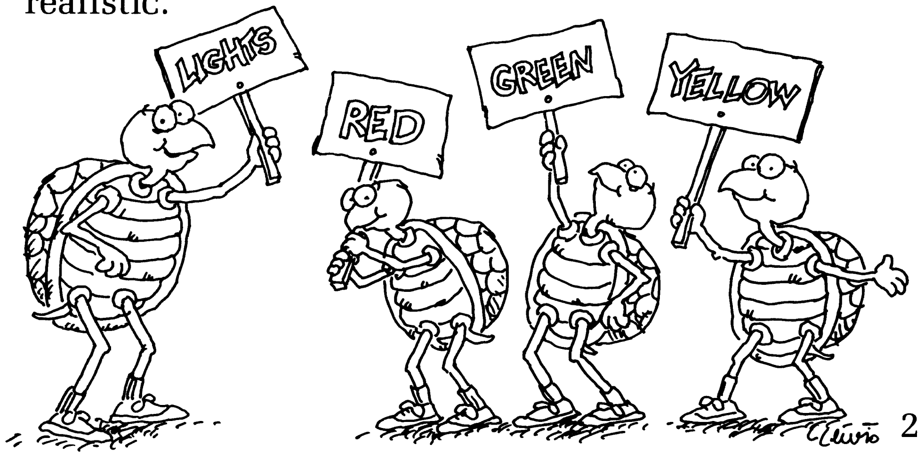
You can also use `green` by itself. Try it!

Challenges

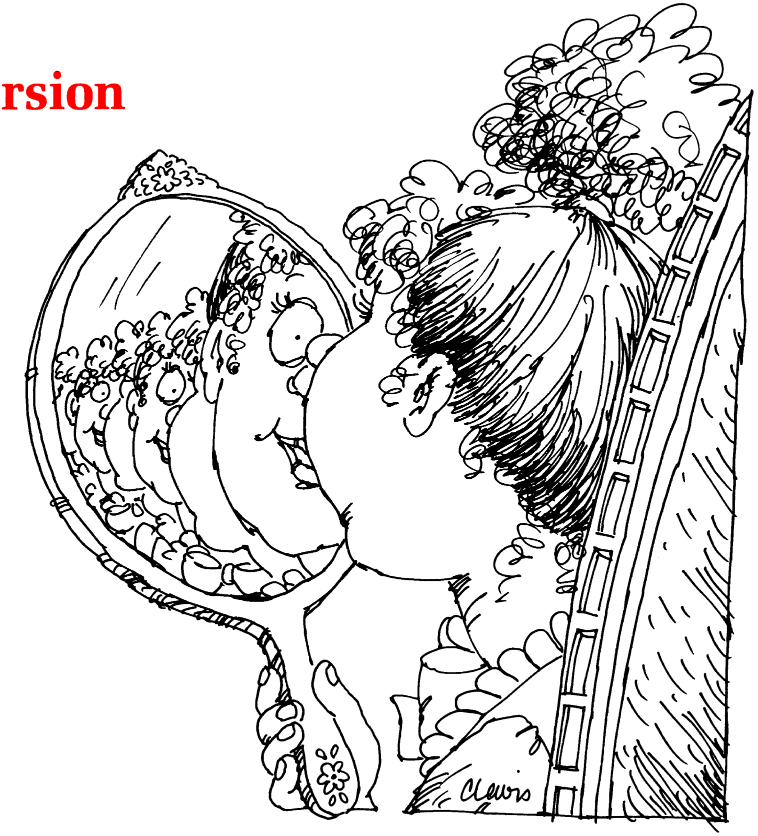
1. Add a procedure named `yellow`. Then change the `lights` procedure so that it makes the lights go Red, then Green, then Yellow.

2. Change the `lights` procedure so that your traffic light goes through the Red-Green-Yellow cycle three times.

3. In a real traffic light, the yellow light does not stay on very long. Why? Change the `yellow` procedure so that the `lights` procedure becomes more realistic.



Recursion



You can make the traffic light repeat the Red-Green-Yellow cycle many times. For example,

```
repeat 4 [red green yellow]
```

will make the light repeat the cycle four times (assuming that you've written procedures for `red`, `green`, and `yellow`).

But what if you want the traffic light to repeat the cycle **forever**? No matter how large a number you use with `repeat`, the traffic light will stop eventually.

But there is a way to make the traffic light continue forever. Simply change the `lights` procedure like this:

```
to lights
red
green
yellow
lights
end
```

◀ This is the important line

When you type `lights` in the Command Center, the computer first runs the `red` procedure. Then it runs `green`, then `yellow`. Then comes the important part: The computer runs the `lights` procedure again!

But when the computer runs the procedure `lights` the second time, it again reaches the line that says `lights`. So the computer runs the `lights` procedure a third time — and a fourth time and a fifth time . . .

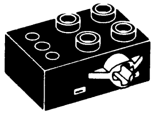
When a procedure includes its own name as part of its definition, it is called a **recursive** procedure.

To stop a recursive procedure, type -S Ctrl-S (**S** is for **S**top). If any lights are still on, type `alloff` (or simply `ao`) to turn them off.

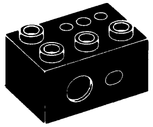
To stop **any** procedure you should type -S Ctrl-S then `alloff` (or `ao`).

Sensors

1. The Touch Sensor



2. The Optosensor



So far, you have used the computer to **send** messages out to LEGO® pieces (motors and lights). But messages can also go in the other direction. The computer can **receive** messages from LEGO® pieces called sensors.

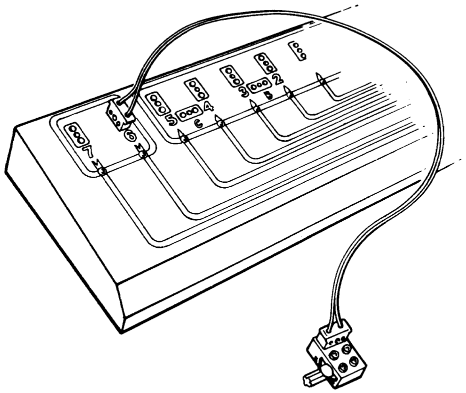
The LEGO® sensors are somewhat like our senses. People have five senses: Sight, smell, hearing, touch, and taste. These senses give us information about what is happening in the world.

LEGO® TC logo has two senses:

The **optosensor** is somewhat like an eye. It tells the computer whether it "sees" light or dark.

The **touch sensor** is somewhat like a finger. It tells the computer whether it "feels" something.

Using a Touch Sensor



You can use a touch sensor to control the traffic light. To start, create a touch sensor by putting a small axle into a touch-sensor brick.

Next, plug the touch sensor into port 6 on the interface box. Then type `listento 6` in the Command Center. This command tells the computer which port to "listen to" to get messages from the sensor.

The sensor will tell the computer whether anything is pressing it. If something is pressing the sensor, the sensor reports `true` to the computer (and the green light under the port goes on). If nothing is pressing the sensor, the sensor reports `false`.

To see what the sensor is reporting to the computer, type `show sensor?` in the Command Center.

Try `show sensor?` several times.

Push-Button Control

You are now ready to use the touch sensor to control the traffic light.

Make sure that the traffic light is connected as it was before: Red in port 0, Yellow in port 1, Green in port 2. Also, make sure the touch sensor is plugged into port 6. Then write this procedure:

```
to switch
listento 6
waituntil [sensor?]
talkto 0
on
end
```

`waituntil` is a LEGO® TC logo primitive that tells the computer to "wait until" the expression in the square brackets reports `true`.

In this case, the computer waits until `sensor?` reports `true` (that is, it waits until you press the sensor). Then the computer turns `on` the red light.

Give it a try. Type `switch`. When you press the sensor, the red light should turn on. (If you change `talkto 0` to `talkto 2` in the `switch` procedure, the green light will go on when you press the sensor.)

Now try this procedure:

```
to button
listento 6
waituntil [sensor?]     ◀ Wait until sensor
                          pressed
talkto 0
on                      ◀ Turn the light on
waituntil               ◀ Wait until release sensor
 [not sensor?]
off                     ◀ Turn the light off
button
end
```

The line `waituntil [not sensor?]` is a bit tricky.

`not sensor?` is the opposite of `sensor?`. It reports `false` when you are pressing the sensor. It reports `true` when you are **not** pressing the sensor.

`waituntil [not sensor?]` tells the computer to wait until `not sensor?` reports `true`. That happens when you are **not** pressing the sensor.

`button` is a recursive procedure, so the light will go on again if you press the sensor again.

Using Two Sensors

You can plug sensors into either port 6 or port 7 — or both.

The computer starts by "listening to" both ports. If you want to use port 7, you must type `listento 7` (or simply `lto 7`). The computer will continue to "listen to" port 7 until the next `listento` command.

Here's a procedure that uses both ports:

```
to double
listento 6                ◀ Listen to port 6
waituntil [sensor?]       ◀ Wait until sensor is
                            pressed

talkto 0
on                        ◀ Turn on the light
listento 7                ◀ Listen to port 7
waituntil [sensor?]       ◀ Wait until sensor is
                            pressed

off                       ◀ Turn off the light
double
end
```

Plug a light into port 0. Then run the `double` procedure. Whenever you press the sensor in port 6, the light will go on. Whenever you press the sensor in port 7, the light will go out.

BUG BOX

You left a space in `waituntil` (It's one word!).
You didn't use square brackets.
The computer is listening to the wrong port.

Challenges

1. What do these procedures do?

```
to three.sec
listento 6
waituntil [sensor?]
talkto 0
onfor 30
three.sec
end

to check
listento 6
show sensor?
check
end
```

Remember: To stop a recursive procedure, press 🍎-S Ctrl-S.

2. Build a LEGO® birthday cake using lights as candles. Write a procedure that makes the lights flash whenever you press the touch sensor.

Other Approaches

There is usually more than one way to write a procedure. For example, `button` (page 31) can be written (at least) two other ways:

```
to button2
listento 6
talkto 0
if sensor? [on]
if not sensor? [off]
button2
end
```

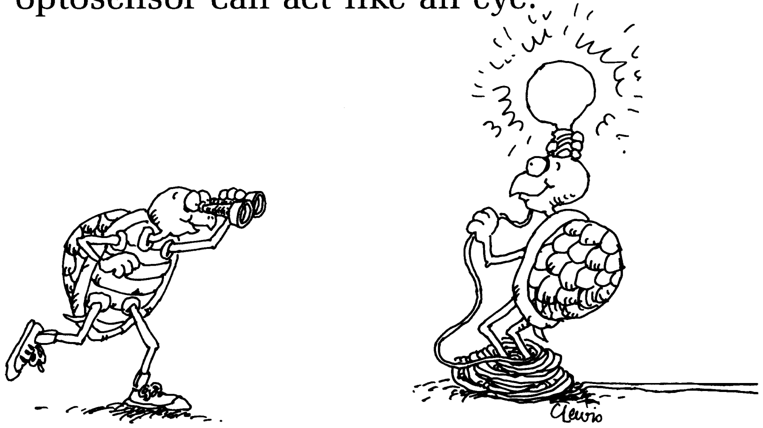
```
to button3
listento 6
talkto 0
ifelse sensor? [on] [off]
button3
end
```

These procedures use the primitives `if` and `ifelse`. `ifelse` covers two cases in one line. If `sensor?` reports `true`, the computer runs the first list (`on`). If `sensor?` reports `false`, the computer runs the second list (`off`).

All three `button` procedures do the same thing: They turn the light on when you push the sensor, and turn the light off when you release the sensor. Which approach is best? Sometimes `waituntil` seems easier, sometimes `if` or `ifelse` seems easier. You should use whichever one seems easiest to you.

Project 4: Electric Eye

Just as the touch sensor can act like a finger, the LEGO® optosensor can act like an eye.



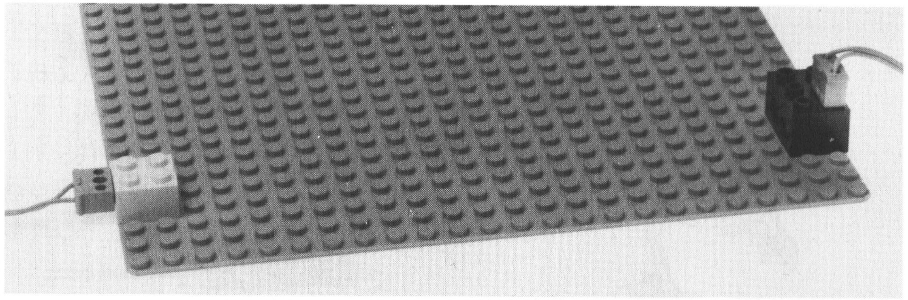
Of course, the optosensor's "vision" is not nearly as good as your vision. Your eyes can see shapes and colors and shades of brightness. The optosensor can only tell the difference between "dark" and "light."

Still, the LEGO® optosensor can be very useful. Imagine an optosensor on one side of a doorway and a LEGO® light on the other side. When nothing is in the doorway, the optosensor sees "light." But when someone moves through the doorway (and blocks the light), the optosensor sees "dark."

You can write many programs using the optosensor. Your program could sound an alarm if anyone walks through the door. Or it could count how many people walk through the door. Or it could measure how long it takes someone to walk through the door.

Using the Optosensor

Like the touch sensor, the optosensor always reports either `true` or `false` to the computer. The optosensor actually acts like a "dark" sensor. When it sees "dark", it reports `true`. When it sees light, it reports `false`.



Set up a LEGO® light across from an optosensor on a flat LEGO building plate, as shown above. The light and sensor should be about 20 studs apart. Plug the light into port 1 and the optosensor into port 6. Now move the cursor to the top part of the page (-U Ctrl-U) and type in this procedure:

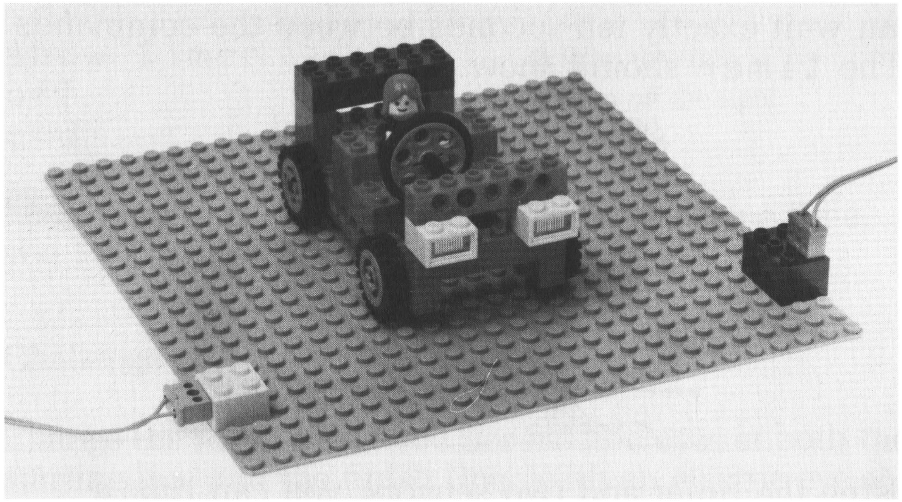
```
to burglar
listento 6
waituntil [sensor?]        ◀ Wait until sensor sees
                             "dark"
show "Burglar!!            ◀ Print out a message
end
```

Return to the Command Center. Type `talkto 1 on` to turn on the light. Then try `burglar`. When you block the light, the computer should print the message.

If you put `talkto 1 on` inside the `burglar` procedure the procedure might not work, since the computer might check `sensor?` before the light has had time to fully "light up."

BUG BOX

You forgot to turn on the light.
The sensor is in the wrong port.
The LEGO® light is too far from the sensor.
There is too much light in the room.
The "burglar" isn't blocking the light completely.



Challenge

Write a procedure that turns on a car motor, but makes the motor stop if the car passes between the light and the sensor.

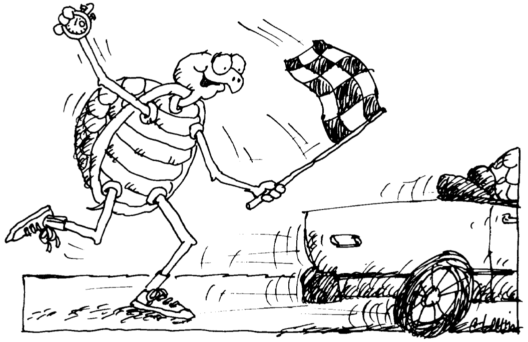
Using the Timer

You can use sensors to start or stop the LEGO®TC logo timer.

To reset the timer to zero, type `resett` in the Command Center (for `reset t`imer). Wait a few seconds then type `show timer`.

Notice that the timer counts in tenths of seconds. So in one second, the timer counts to 10. In two seconds, it counts to 20.

Try `resett` and `show timer` again. See if you can wait exactly ten seconds between the commands. (The `timer` should show `100`.)



Using the timer and two sensors, you can time a Soap Box Derby race. Use the soap box car you built earlier or build a new one.

Plug the touch sensor into port 7. This sensor can act as the "starting gun." The timer should start when you press the sensor.

You can use the electric eye as the finish line. (Set it up as described on page 36.) The timer should stop when the car passes in front of the electric eye.

Your procedure for the Soap Box timer should look something like this:

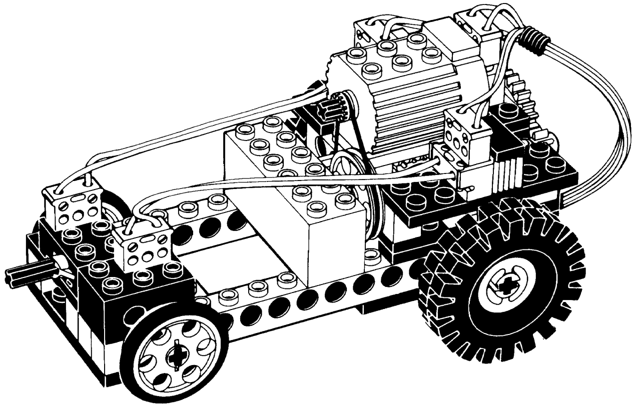
```
to time.race
talkto 1
on                     ◀ Turn on the light
listento 7             ◀ Listen to the touch sensor
waituntil [sensor?]    ◀ Wait until sensor is pressed
resett                 ◀ Start the timer
listento 6             ◀ Listen to the optosensor
waituntil [sensor?]    ◀ Wait until sensor "sees"
                         car
show timer             ◀ Print the time
off                    ◀ Turn off the light
end
```

Make sure to push the touch sensor at the same time you let the Soap Box car go!

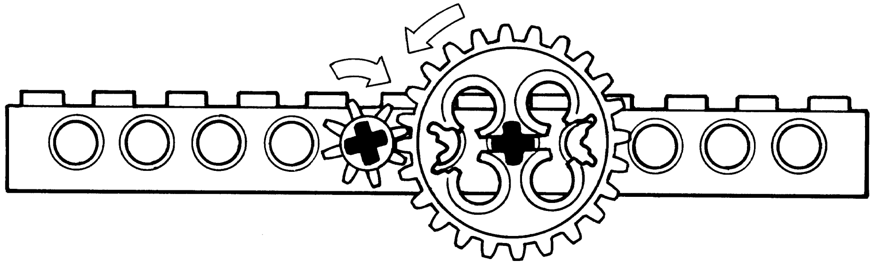
Challenge

Change the race so that you use electric eyes at both the starting line and the finish line. With an electric eye at the starting line, you do not have to worry about pushing the touch sensor at the beginning of the race. Do you have to change the procedure `time.race`?

Project 5: "Smart" Cars



The car you built from Activity Card 3 has four gears and two pulleys. What is the advantage of using so many gears? One reason for using gears is to make a machine **faster** or **slower**.

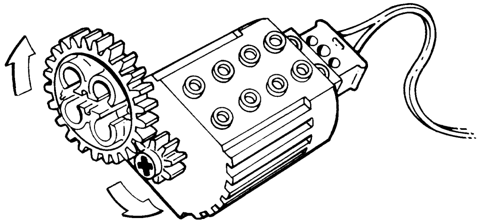


A combination of gears is called a **gear train**. To build the one shown above, place one axle through an 8-tooth gear and another axle through a 24-tooth gear. Then put the axles through the holes on a 12-stud beam. Make sure the teeth of the gears are meshed together properly. Spin one of the axles. The small gear will turn **faster**. Turn the big gear around once and count how many times the small gear turns.

Although the big gear turns more slowly, it is **stronger**. It would be better for driving a car up a hill, or lifting a heavy object.

There is a **tradeoff** between speed and strength. When a small gear and a big gear are meshed together, the small gear is faster and weaker. The big gear is slower and stronger.

You can feel this tradeoff when you ride a ten-speed bicycle. In high gear, you can go fast, but pedaling is difficult. In low gear, you go more slowly, but pedaling is easier.

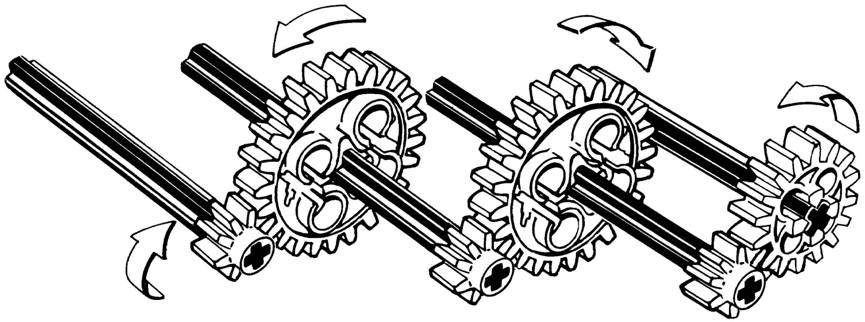


By using LEGO® gears, you can change the speed and strength of a LEGO® car. LEGO® motors are very fast but very weak. If you connect a motor directly to the wheel of a car, it might not even have enough strength to make the car move.

You can make the car stronger (and slower) by using a **small-to-big** combination of gears, as shown above.

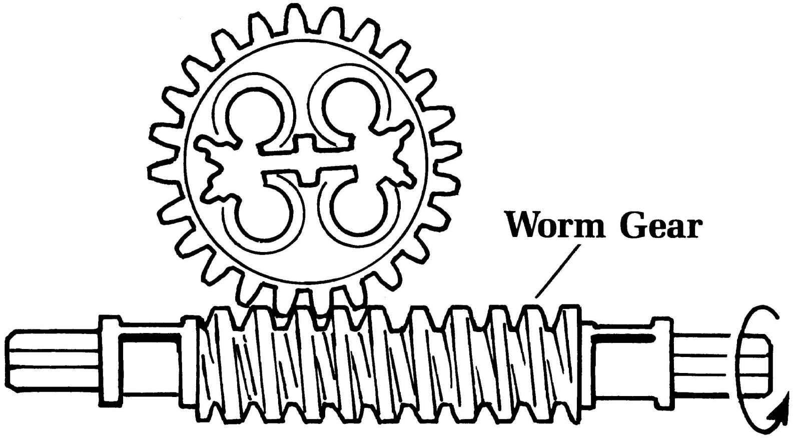
If the big gear has three times as many teeth as the small gear, then the wheel will be three times stronger than the motor. But the motor will turn three times faster.

To make an even bigger change you can build a gear train with more than two gears.



Every time gears "mesh" together, the speed and strength changes. When it goes small to big, it becomes slower and stronger. When it goes big to small, it becomes faster and weaker.

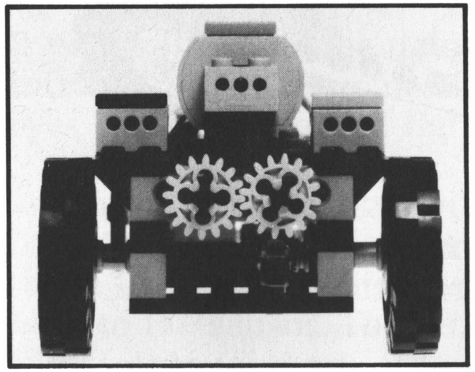
Worm gears are particularly useful in gear trains. When a worm gear is meshed with another gear, the worm gear is always faster and weaker. So the worm gear acts like a very "small" gear. (In fact, it acts like a gear with only one tooth!)

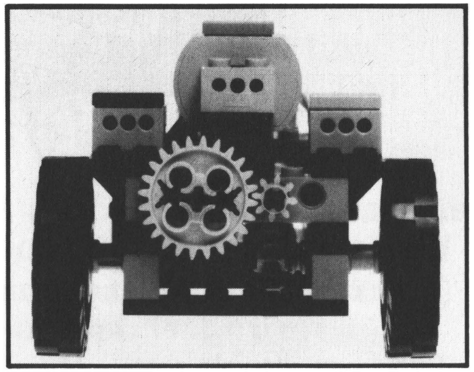


Look at the gear train on your car. Note that every combination of "meshed" gears goes from small to big. Thus, the gear train makes the car much slower and stronger than the motor itself.

Challenge

You can change the gears on the LEGO® car you just built. Here are two possibilities:



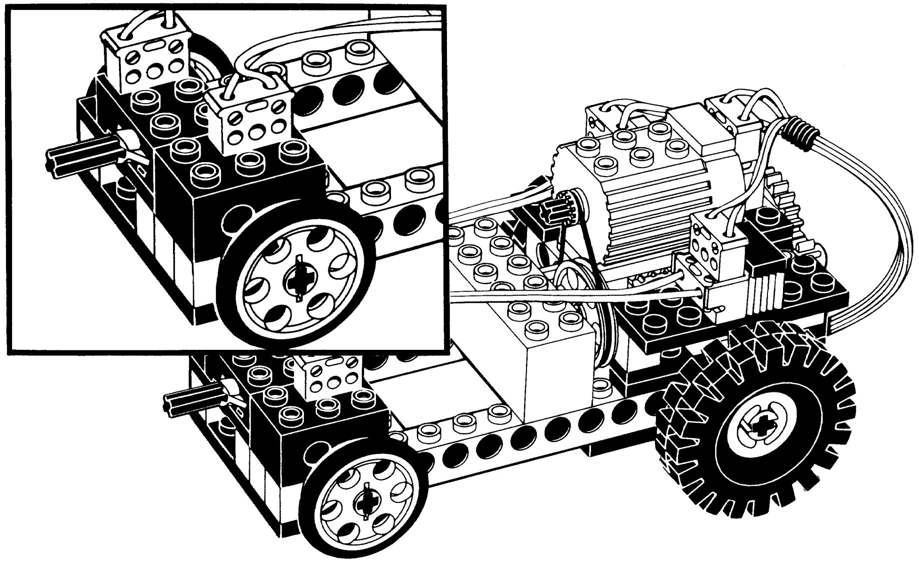


For each combination, test how far the car goes in five seconds (use `onfor 50`).

Which one goes the furthest? Why?

Which one would be best in a tug-of-war? Why?

Sensory Car



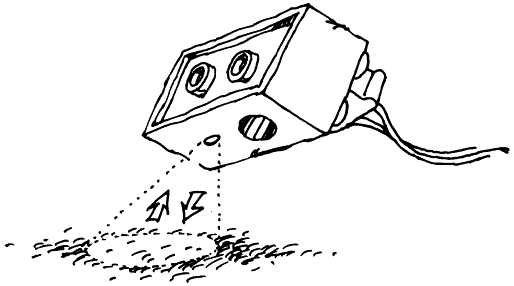
If you add sensors to your car, the car can receive (and react to) information from the world around it.

To start, attach a touch sensor to the front of the car, as shown above. You can write a procedure that makes the car stop when it bumps into something:

```
to collide
talkto "A
on
listento 6
waituntil [sensor?]
off
end
```

Make sure that the motor and sensor are plugged into the correct ports. Then try `collide`.

You could use an optosensor instead of the touch sensor. The optosensor can tell the difference between white and black. So you can program the car to react when it "sees" a black line on a white surface.

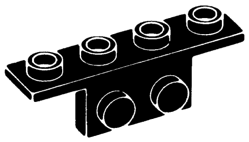


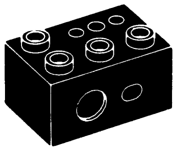
It works like this: the optosensor brick has a tiny **infrared** light that comes from the sensor slot. Our eyes can't see infrared light, but the sensor can.

When the infrared light hits a white surface, it reflects back to the sensor. The sensor reports `false`. When the light hits a black surface, not much light reflects back to the sensor. The sensor reports `true`. (Remember: The optosensor acts like a "dark sensor." It reports **true** when it "sees" dark.)

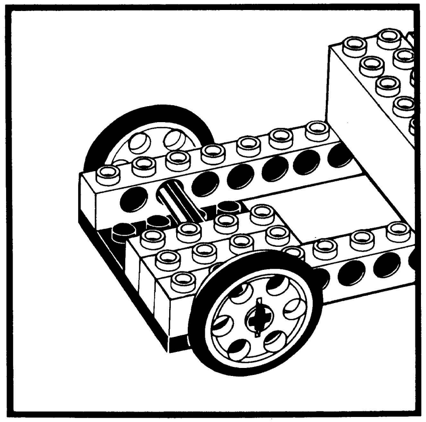
For this work, the sensor should be about three millimeters from the surface (about the thickness of a LEGO® plate).

To attach the optosensor to the front of the car you will need:

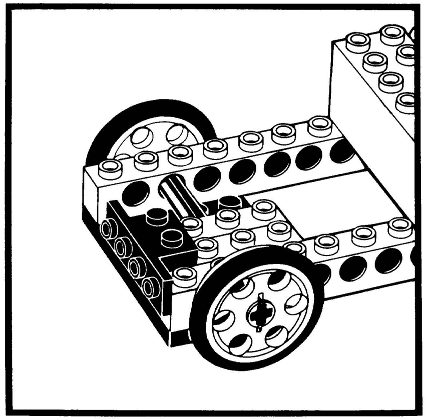


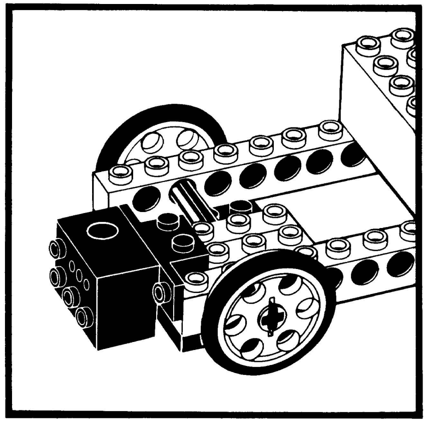


Take the optosensor and touch sensor off the front of the car. Also remove the 2 × 4 plate that the touch sensor is attached to. Your car will look like this:



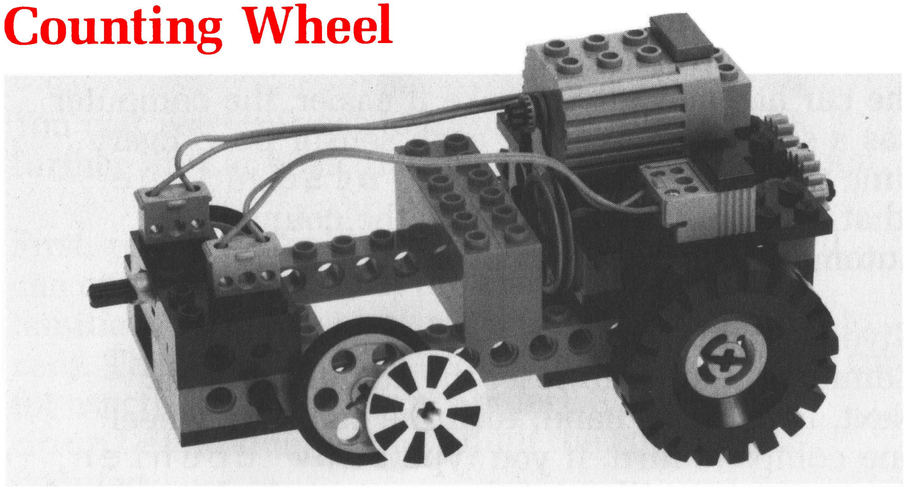
Then build:





Place a sheet of white paper on the floor. Place a sheet of black paper next to the white sheet, overlapping the edges. Put your car onto the white sheet. Plug the optosensor into port 6. Run the `collide` procedure.

Counting Wheel



By using an optosensor and a LEGO® counting wheel, you can keep track of exactly how far your car travels.

The counting wheel has black and white slices. On one side, there are four slices of each color. On the other side, there are eight.

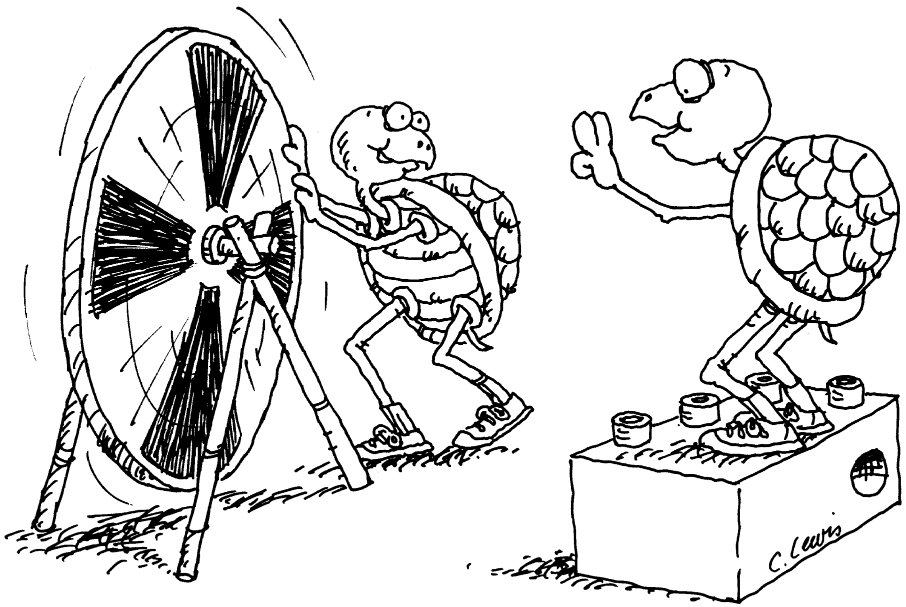
When the sensor is next to a black "slice," it reports `true`. When it is next to a white "slice," it reports `false`. Turn `on` the car motor, (type `talkto "A on`), then run the `check` procedure:

```
to check
listento 6
show sensor?
check
end
```

Make sure the sensor is plugged into port 6!
Remember: To stop a recursive procedure, press -S
Ctrl-S.

You can use this information to keep track of how far the counting wheel has turned — and how far the car has moved! To make it easier, the computer has a special **counter** for each sensor port. Every time the sensor changes from `false` to `true` (that is, from white to black), the counter **automatically** increases by one!

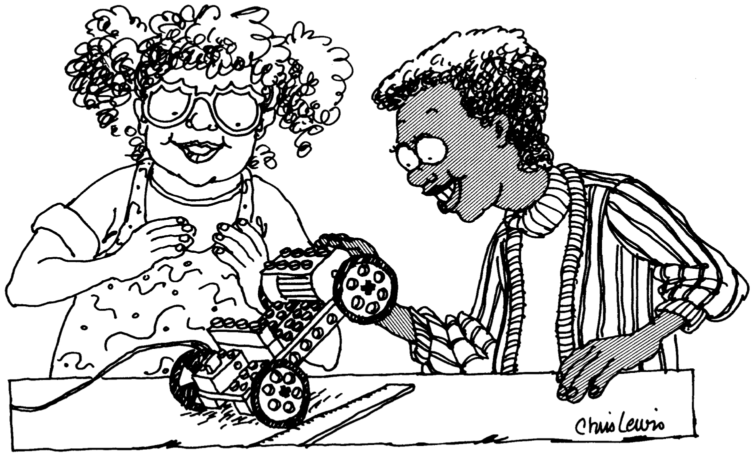
Give it a try. First, type `resetc` in the Command Center to reset the counter to zero. Next, using your hand, turn the counting wheel one complete turn. If you type `show counter`, the computer will print the number of white "slices" on the wheel.



Measuring Distance

You can use your car to measure distance. The further the car goes, the higher the counter counts!

First, you must "calibrate" the counter so that you can convert the value of the counter into more familiar units. Type `resetc` to reset the counter to zero. Then, lift up the back of the car and roll the car for exactly one meter (use a ruler). Type `show counter` to find out how high the counter has counted.

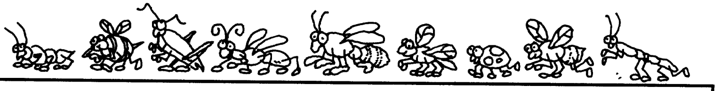


Let's assume that the counter has counted to 60. (You will probably get a different number, but the idea will be the same.)

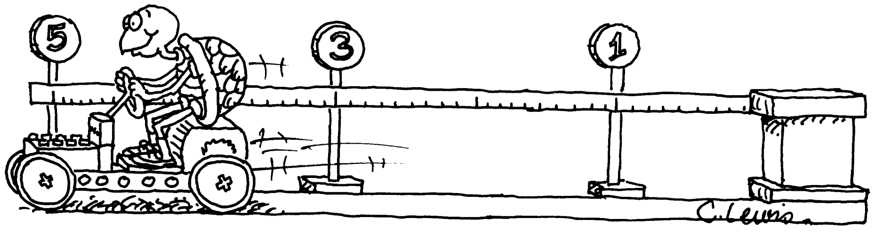
Now you are ready to use the car as a measuring device. Let's say you want to measure the length of a table top. Reset the counter and roll the car from one end of the table to the other. Then check the counter (using `show counter`).

If the counter counted to 30, then the table is one-half meter long (30/60 = 1/2 = 0.5). If the counter counted to 120, then the table is two meters long (120/60 = 2).

BUG BOX



You forgot to reset the counter.
The sensor is plugged into the wrong port.
The sensor is too close to the counting wheel.
The sensor is too far from the counting wheel.



Challenges

1. Write a procedure `showcount` that keeps printing the value of the counter as you roll the car.

2. Change `showcount` to `showmeters` so that the computer prints the number of meters the car has moved. Write another procedure `showcm` that prints the number of centimeters the car has moved. (Use / as the divide symbol and use * as the multiplication symbol.)

3. Use the car to measure distances on a map. Write a procedure `showkm` that prints the number of kilometers the car has moved. (Hint: Look at the scale of the map.)

Advanced Project

Programming by Distance

Now that you can convert between the counter and units of distance, you can control how far the car moves.

Assume that the counter counts to 60 every time the car moves one meter. Then the procedure `onemeter` will make the car move one meter:

```
to onemeter
listento 6
resetc
talkto "A
on
waituntil [counter = 60]
off
end
```

Similarly, you can create a procedure `twometers` by changing one number. Simply change

`[counter = 60]` to `[counter = 120]`

You could keep making other procedures like `threemeters` and `fourmeters`. But it is easier to create one general procedure called `meters`.

The meters procedure should take a number of meters as an **input**. So if you type `meters 3`, the car should move three meters. And if you type `meters 67`, the car should move 67 meters (if your cable is long enough!)

```
to meters :distance
listento 6
resetc
talkto "A
on
waituntil [counter = 60 * :distance]
off
end
```

Notice the colon (:)

Now if you type `meters 5`, the number `5` will replace `:distance` wherever `:distance` appears inside the `meters` procedure. So the key line becomes

```
waituntil [counter = 300]
```

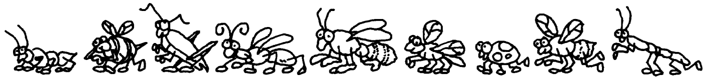
(since 60 * 5 = 300).

Similarly, if you type `meters 2`, the key line becomes

```
waituntil [counter = 120]
```

(since 60 * 2 = 120).

BUG BOX



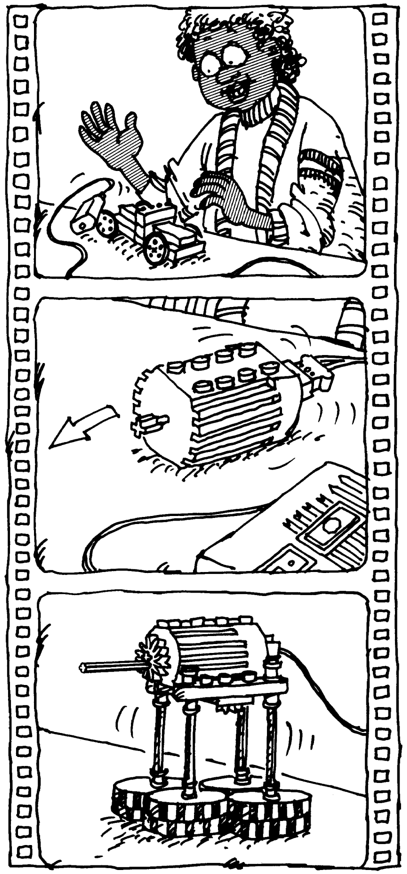
You forgot the colon (`:`).
You forgot to put `:distance` on the first line of `meters`.
You didn't put spaces around the `=` or the `*` .

Challenge

Figure out the speed of your car, both in meters per second and in kilometers per hour.

Invention Theater

Ideas can come from the most unexpected places. Inventors are always on the lookout for things that happen by chance. By connecting unexpected events with things you already know, you can come up with new ideas. These "happy accidents" are called **serendipity**.



One day, Nicky was experimenting with his LEGO® car. Suddenly, the motor fell off.

As it shook across the table, he wondered if he could build a machine that moved by shaking itself, without the use of wheels.

So Nicky built a LEGO® "walker." A rotating arm strengthened the motor vibrations and made the walker move.

You can be an inventor too, just like Nicky.
If you think your invention would make a good story for Invention Theater, write it up and send it to the appropriate address below:

Invention Theater
LEGO Dacta Educational Products
555 Taylor Rd.
Enfield, CT 06082
U.S.A.

Invention Theatre
LEGO Dacta Education Division
P.O. Box 639
Lane Cove
N.S.W. 2066
AUSTRALIA

Glossary List

The page numbers in the definitions refer to the ***Getting Started*** *book pages where the words are first described.*

axle: A thin rod that connects to the middle of a wheel (or gear). The wheel and axle rotate together.

Command Center: The place where you type commands for the computer to carry out. The Command Center is at the bottom of the computer screen (page 11).

cursor: A small blinking light that indicates the current position on the computer screen (page 11).

debugging: The act of finding and fixing things that aren't working right. There might be "bugs" in either your LEGO® machines or your Logo procedures (page 22).

force: A push or pull that causes an object to speed up or slow down. Gravity and friction are examples of forces (page 6).

friction: A force that causes an object to slow down. Rough surfaces cause more friction than smooth surfaces.

gear: A wheel with "teeth." You can use gears to change the position, strength, speed, and direction of a rotating motion (pages 6, 40-43).

gear train: A combination of gears used for changing the strength and speed of a rotating motion. When a small gear "meshes" with a large gear, the large gear rotates more slowly, but with greater strength (page 40).

gravity: A universal force that pulls all objects toward one another. The Earth's gravity pulls all objects down toward the center of the Earth.

infrared: A type of light beyond the red end of the rainbow. The LEGO® optosensor sends out and detects infrared light (page 45).

interface box: The box that connects LEGO® motors, lights, and sensors to the computer. The interface box has 12 ports where you can plug in motors, lights, and sensors (page 9).

list: A special type of Logo object that combines several pieces of information into a single unit. Lists are enclosed in square brackets, like `[1 2 3]`. By using a list, you can manipulate many pieces of information at once, rather than separately (page 17).

motor: A machine that provides a rotating motion. LEGO® motors convert electrical energy to mechanical energy (page 6).

optosensor: A sensor that detects changes in light. The LEGO optosensor usually acts like a "dark sensor." It reports `true` when it "sees" something dark (page 28).

port: A set of holes in the interface box where you can plug a LEGO® wire. Some ports (**outlets**) send messages to motors and lights. Other ports (**inlets**) receive messages from sensors (page 9).

primitive: A procedure that is already built into the Logo language (page 21).

procedure: A named set of Logo instructions. When you type the name of a procedure in the Command Center, the computer runs the procedure (page 20).

Procedure Center: The place where you define procedures. The Procedure Center is at the top of the computer screen (page 21).

recursive procedure: A procedure that includes itself as part of its definition. Some recursive procedures run through the same "loop" over and over (page 26).

sensor: A device that sends information about the world to the computer. LEGO® TC logo has touch sensors and optosensors (page 8).

software: Instructions that tell the computer what to do. Software is often stored on a **disk**.

subprocedure: A procedure that is used within the definition of another procedure (page 24).

theory: A set of beliefs or guesses about the way things work (page 5).

LEGO® TC logo
LEGO Dacta Educational Products
555 Taylor Road
Enfield, CT 06082

LEGO Australia Pty. Ltd.
P.O. Box 639
Lane Cove
N.S.W. 2066

ISBN 0-914831-68-2
198323

Printed in U.S.A.